Parle Tilak Vidyalaya Associations

SATHAYE COLLEGE (Autonomous)

Vile-Parle (East), Mumbai – 400 057.

Practical Journal

Blockchain

Submitted by

Vaishnavi Kasare

Seat No.: **15**

M.Sc. [I.T.]-Information Technology Part II

2022 – 2023

Parle Tilak Vidyalaya Association's

SATHAYE COLLEGE (Autonomous)

Vile-Parle (East), Mumbai – 400 057.

CERTIFICATE

This is to certify that Vaishnavi Vishnu Kasare

Seat No 15 has successfully completed all the practicals in the subject of Blockchain for M.Sc.I.T. Part-II SEM – IV as prescribed by University of Mumbai for the year 2022-2023.

Coordinator Professor in Charge External Examiner
M.Sc. [I.T.]

Date: Date Date

INDEX

6	**31.04.2023**	**Practical No 6** Write a solidity program to find modulus of addition of two numbers with DD and modulus of multiplication of two numbers with MM where the two numbers are achieved by performing AND operation of DD and MM of your date of birth and OR operation of YY and YY from year of birth.	
7	**04.05.2023**	**Practical No 7** **a)** Create a smart contract to demonstrate Mathematical function. **b)** Create a smart contract to demonstrate Function overloading.	
8	**08.05.2023**	**Practical No 8** **a)** Create a smart contract to show the implementation of Interface. **b)** Create a smart contract to show the implementation of Inheritance.	
9	**10.05.2023**	**Practical No 9** Write a solidity program to create an array of roll numbers and then create a smart contract where it checks the value of the roll number and perform AND operation with today's date and if the result is even then allow the student else deny (DD part only).	
10	**11.05.2023**	**Practical No 10** Write a solidity program to find the sum of an array of ten numbers using loop the numbers are expected to be taken from the user, create a smart contract to find the AND operation of odd positioned numbers and OR operation of even positioned numbers including 0th index. Hence find the product of the results and also identify whether the result is the part of array or not.	

Practical No 1

Aim: Create blockchain with 3 blocks and hence display the entire blockchain, hash value and timestamp of each block.

Code:

File Edit Format Run Options Window Help

```
import datetime
import hashlib

class Block:
    def __init__(self, previous_block_hash, data, timestamp):
        self.previous_block_hash = previous_block_hash
        self.data = data
        self.timestamp = timestamp
        self.hash = self.get_hash()

    @staticmethod
    def create_genesis_block():
        return Block("0", "0", datetime.datetime.now())

    def get_hash(self):
        header = (str(self.previous_block_hash) + str(self.data) + str(self.timestamp))
        inner_hash = hashlib.sha256(header.encode()).hexdigest().encode()
        comp_hash = hashlib.sha256(inner_hash).hexdigest()
        return comp_hash

number_of_blocks = 3

Blockchain = [Block.create_genesis_block()]
print("Genesis block is created")
print("Hash: %s" % Blockchain[0].hash)

for i in range(1, number_of_blocks):
      Blockchain.append(Block(Blockchain[i-1].hash, "Block number %d" %i, datetime.datetime.now()))
      print("%d block created" % i)
      print("Hash: %s" % Blockchain[-1].hash)
      print("timestamp: ",datetime.datetime.now())
```

Output:

IDLE Shell 3.10.0

File Edit Shell Debug Options Window Help

```
Python 3.10.0 (tags/v3.10.0:b494f59, Oct  4 2021, 19:00:18) [MSC v.1929
AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
= RESTART: C:/Users/mazhar/AppData/Local/Programs/Python/Python310/block
va quesion.py
Genesis block is created
Hash: 80069ac29e94bdbfd44d7dcef3be07daee116e7be746171439000fa308335637
1 block created
Hash: 48d63604a1f7e3f2ec4cef8ecb9c67a65bff3c18acc1109d9e9ab5397f326010
timestamp:  2023-05-09 18:25:59.018541
2 block created
Hash: 2f54e5cccd292057f5aaeb14d28054763fd91d34bb69dc50805c56ca65dc6ab4
timestamp:  2023-05-09 18:25:59.049793
```

Practical No 2

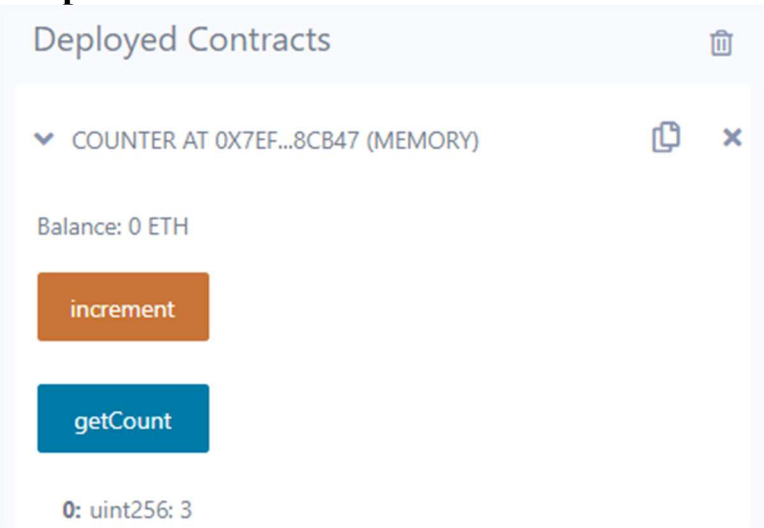
2a) Aim: Create a smart contract for counter.

Code:

Home | 1_Storage.sol | 2_a_counter.sol

```
// SPDX-License-Identifier: GPL-3.0

pragma solidity >=0.8.2 <0.9.0;

contract Counter {

    uint256 private count=0;

    function getCount() public view returns (uint256) {
        return count;
    }

    function increment() public {    infinite gas
        count += 1;
    }
}
```

Output:



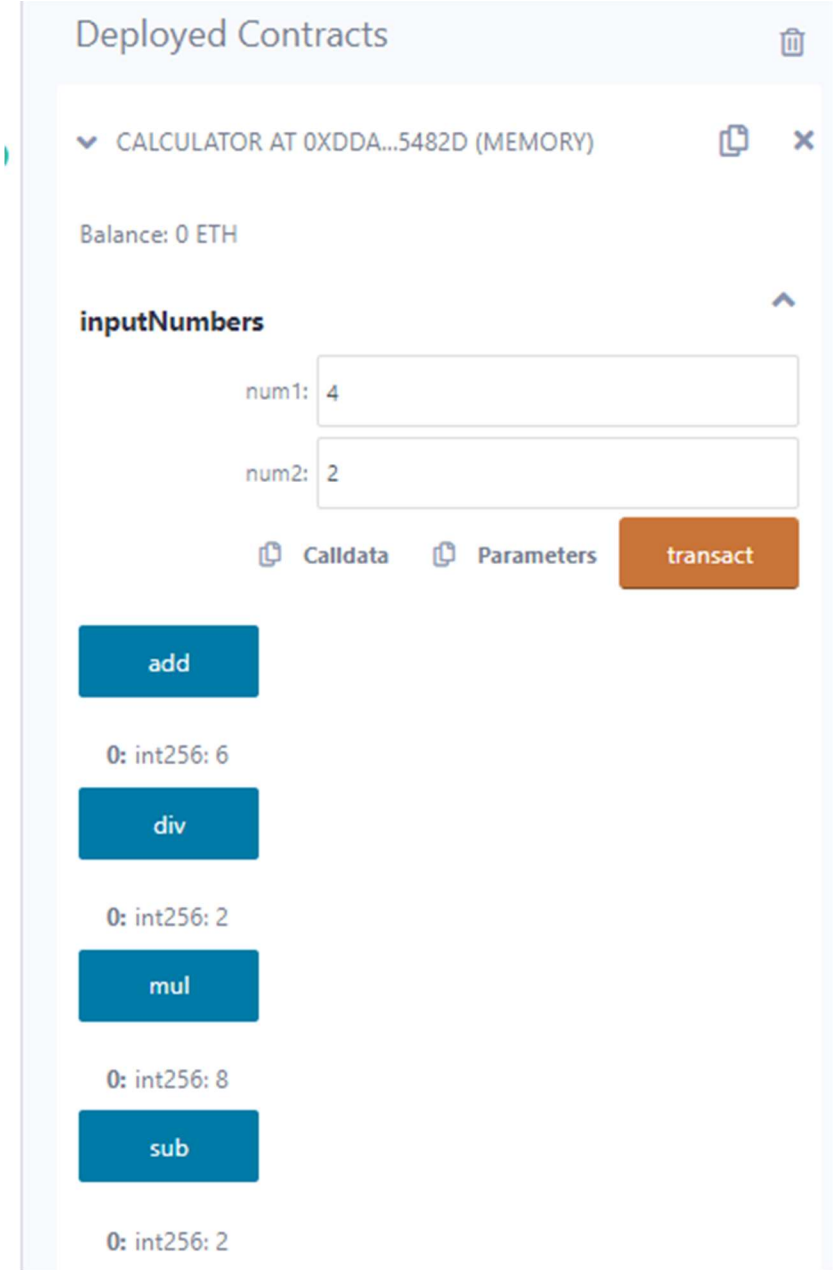
2b) Aim: Create a smart contract for calculator.

Code:

2_b_calculator.sol

```
// SPDX-License-Identifier: GPL-3.0

pragma solidity >=0.8.2 <0.9.0;

contract Calculator {

    int256 number1;
    int256 number2;

    function inputNumbers(int256 num1, int256 num2) public {    infinite gas
        number1 = num1;
        number2 = num2;
    }

    function add() public view returns (int256){    infinite gas
        int256 result = number1 + number2;
        return result;

    }
    function sub() public view returns (int256){    infinite gas
        int256 result = number1 - number2;
        return result;

    }
    function mul() public view returns (int256){    infinite gas
        int256 result = number1 * number2;
        return result;

    }
    function div() public view returns (int256){    infinite gas
        int256 result = number1 / number2;
        return result;

    }
}
```

Output:


Deployed Contracts
CALCULATOR AT 0XDDA...5482D (MEMORY)
Balance: 0 ETH
inputNumbers
num1: 4
num2: 2
Calldata
Parameters
transact
add
0: int256: 6
div
0: int256: 2
mul
0: int256: 8
sub
0: int256: 2

Practical No 3

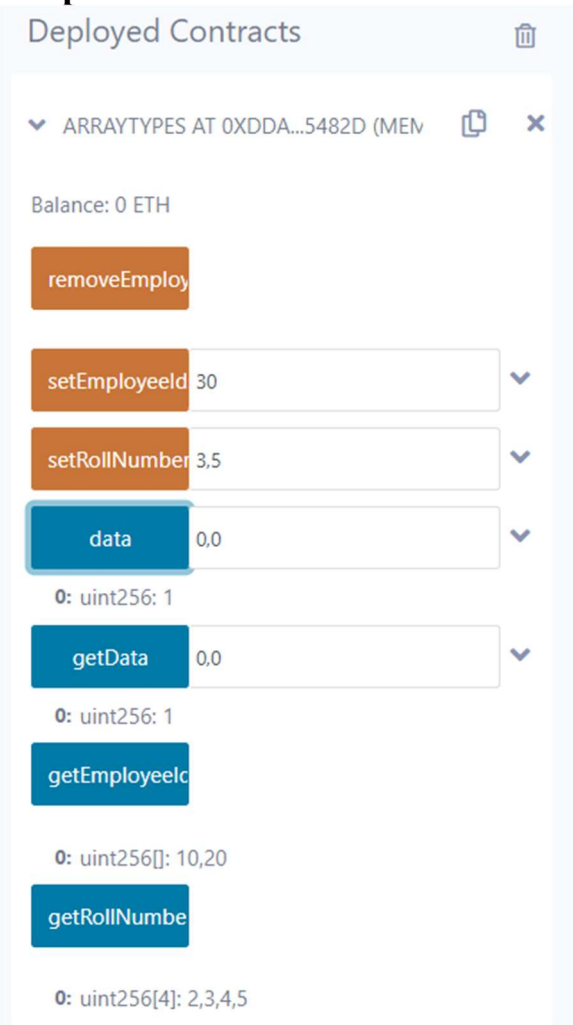
3a) Aim: Write a solidity program to demonstrate array and its types.

Code:

3_a_arrays.sol

```
// SPDX-License-Identifier: GPL-3.0

pragma solidity >=0.8.2 <0.9.0;

contract ArrayTypes {

    //< ==================== Fixed Size Array ====================>
    uint[4] rollNumbers;

    function setRollNumbers(uint _index, uint _rollNumber) public {    infinite gas

        rollNumbers[_index] = _rollNumber;
    }

    function getRollNumbers() public view returns(uint[4] memory) {    infinite gas
        return rollNumbers;
    }

    //< ==================== Dynamic Size Array ====================>
    uint[] employeeIds;

    function setEmployeeIds(uint _empId) public {    46873 gas
        employeeIds.push(_empId);
    }

    function removeEmployeeId() public {    29511 gas
        employeeIds.pop();
    }

    function getEmployeeIds() public view returns(uint[] memory) {    infinite gas
        return employeeIds;
    }

    //< ==================== Multi-Dimensional Array ====================>

    // uint256[col][row] array_name
    uint[2][3] public data = [[1,2], [3,4],[5,6]];

    function getData(uint _col, uint _row) public view returns (uint) {    infin
        uint number = data[_col][_row];
        return number;
    }
}
```

Output:


Deployed Contracts
ARRAYTYPES AT 0XDDA...5482D (MEM
Balance: 0 ETH
removeEmploy
setEmployeeId
30
setRollNumber
3,5
data
0,0
0: uint256: 1
getData
0,0
0: uint256: 1
getEmployeeIc
0: uint256[]: 10,20
getRollNumbe
0: uint256[4]: 2,3,4,5

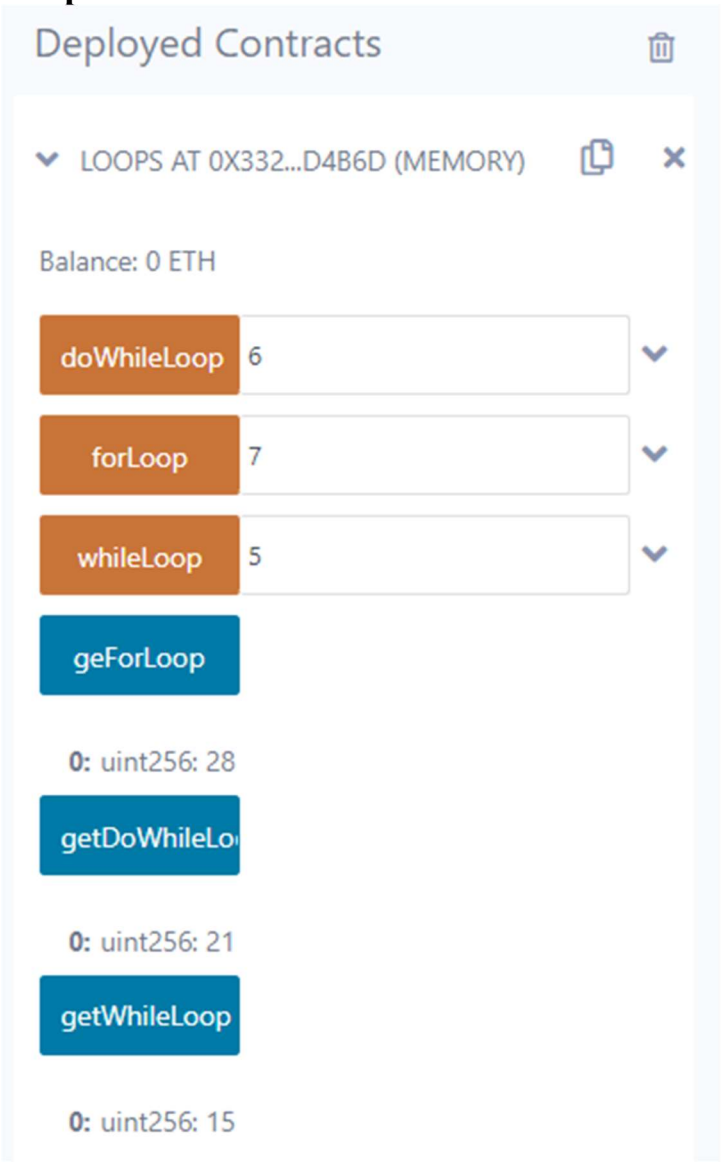
3b) Aim: Demonstrate the use of loops in solidity.

Code:

Home | 3_b_loops.sol

```
// SPDX-License-Identifier: GPL-3.0

pragma solidity ^0.8.0;

contract Loops {

    // while loop
    uint256 private result=0;
    function whileLoop(uint256 number) public returns (uint256) {    infinite gas
        uint256 i = 1;
        while (i <= number) {
            result += i;
            i++;
        }
        return result;
    }

    function getWhileLoop() public view returns (uint256) {    2503 gas
        return result;
    }

    // do while
    uint256 private result1=0;
    function doWhileLoop(uint256 number) public returns(uint256) {    infinite gas
        uint256 i = 1;
        do {
            result1 += i;
            i++;
        } while(i <= number);
        return result1;
    }

    function getDoWhileLoopResult() public view returns (uint256) {    2437 gas
        return result1;
    }


    // for loop
    uint256 private result2=0;

    function forLoop(uint256 number) public returns(uint256) {    infinite gas
        for(uint256 i=1;i<=number;i++) {
            result2 +=i;
        }
        return result2;
    }

        function geForLoop() public view returns (uint256) {    2415 gas
        return result2;
    }

}
```

Output:


Deployed Contracts
LOOPS AT 0X332...D4B6D (MEMORY)
Balance: 0 ETH
doWhileLoop
6
forLoop
7
whileLoop
5
geForLoop
0: uint256: 28
getDoWhileLo
0: uint256: 21
getWhileLoop
0: uint256: 15

Practical No 4

Aim: Write a solidity program to demonstrate the use of following operators:

a) Comparison b) Logical c) Assignment d) Ternary e) bitwise

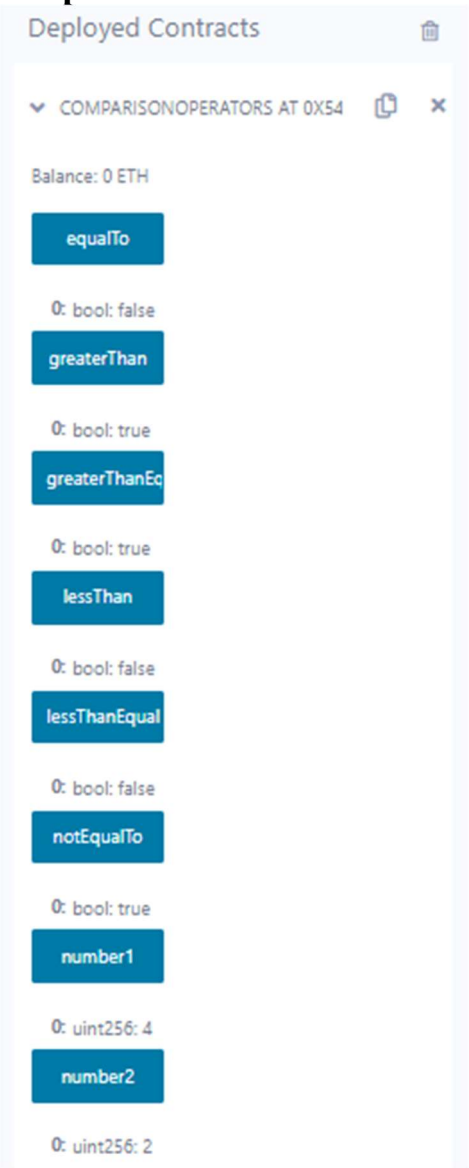
4a) Aim: Solidity program to demonstrate Comparison operators.

Code:

4_b_LogicalOperators.sol 2 | 4_a_ComparisonOperators.sol

```
// SPDX-License-Identifier: GPL-3.0

pragma solidity >=0.8.2 <0.9.0;

contract ComparisonOperators {

    uint public number1 = 4;
    uint public number2 = 2;

    // ==
    function equalTo() public view returns (bool) {    4584 gas
        bool result = number1 == number2;
        return result;
    }

    // !=
    function notEqualTo() public view returns (bool) {    4610 gas
        bool result = number1 != number2;
        return result;
    }

    // <
    function lessThan() public view returns (bool) {    4563 gas
        bool result = number1 < number2;
        return result;
    }

    // <=
    function lessThanEqual() public view returns (bool) {    4609 gas
        bool result = number1 <= number2;
        return result;
    }

    // >
    function greaterThan() public view returns (bool) {    4628 gas
        bool result = number1 > number2;
        return result;
    }

    // >=
    function greaterThanEqual() public view returns (bool) {    4632 gas
        bool result = number1 >= number2;
        return result;
    }
}
```

Output:


Deployed Contracts
COMPARISONOPERATORS AT 0X54
Balance: 0 ETH
equalTo
0: bool: false
greaterThan
0: bool: true
greaterThanEq
0: bool: true
lessThan
0: bool: false
lessThanEqual
0: bool: false
notEqualTo
0: bool: true
number1
0: uint256: 4
number2
0: uint256: 2

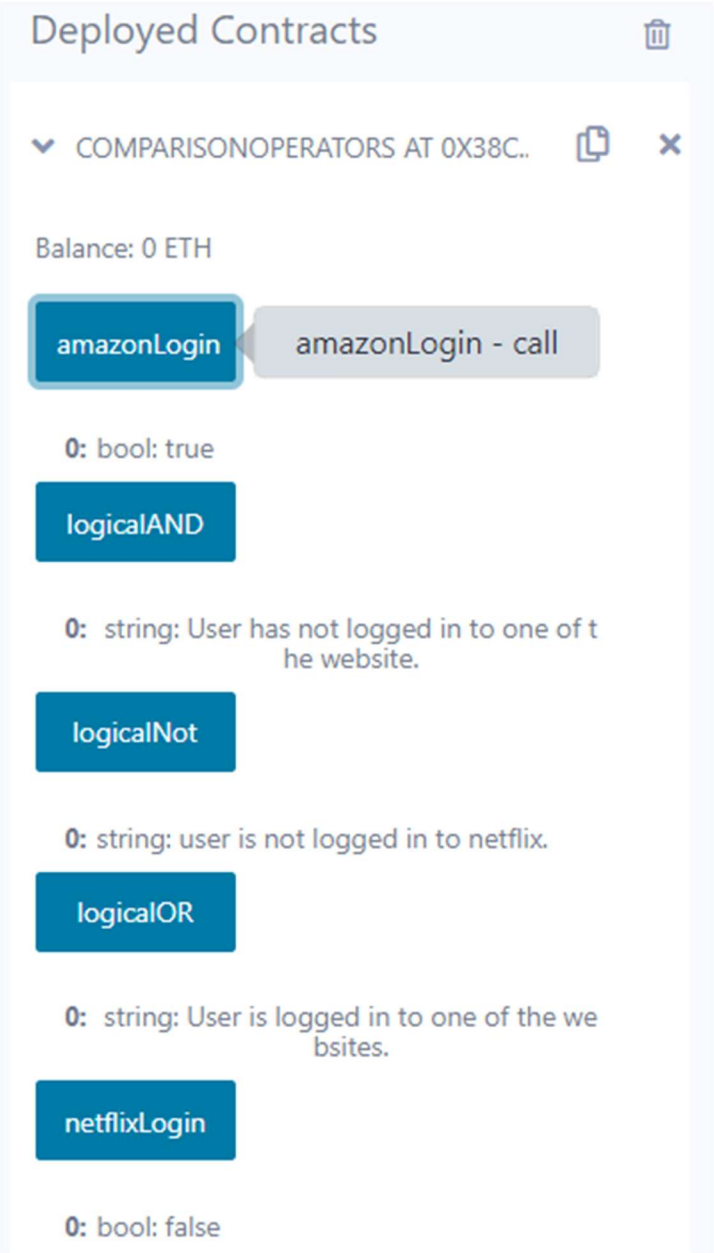
4b) Aim: Solidity program to demonstrate Logical operators.

Code:

4_b_LogicalOperators.sol

```
// SPDX-License-Identifier: GPL-3.0

pragma solidity >=0.8.2 <0.9.0;

contract ComparisonOperators {

    bool public amazonLogin = true;
    bool public netflixLogin = false;

    // !
    function logicalNot() public view returns (string memory) {     infinite gas
        string memory result;
        if(!netflixLogin) {
            result = "user is not logged in to netflix.";
        } else {
            result = "user is logged in to netflix.";
        }
        return result;
    }

    // &&
    function logicalAND() public view returns (string memory) {     infinite gas
        string memory result;
        if(netflixLogin && amazonLogin) {
            result = "User has logged in to both websites.";
        } else {
            result = "User has not logged in to one of the website.";
        }
        return result;
    }

    // ||
    function logicalOR() public view returns (string memory) {     infinite gas
        string memory result;
        if(netflixLogin || amazonLogin) {
            result = "User is logged in to one of the websites.";
        } else {
            result = "User is not logged in to both of the websites.";
        }
        return result;
    }
}
```

Output:


Deployed Contracts
COMPARISONOPERATORS AT 0X38C..
Balance: 0 ETH
amazonLogin
amazonLogin - call
0: bool: true
logicalAND
0: string: User has not logged in to one of t he website.
logicalNot
0: string: user is not logged in to netflix.
logicalOR
0: string: User is logged in to one of the we bsites.
netflixLogin
0: bool: false

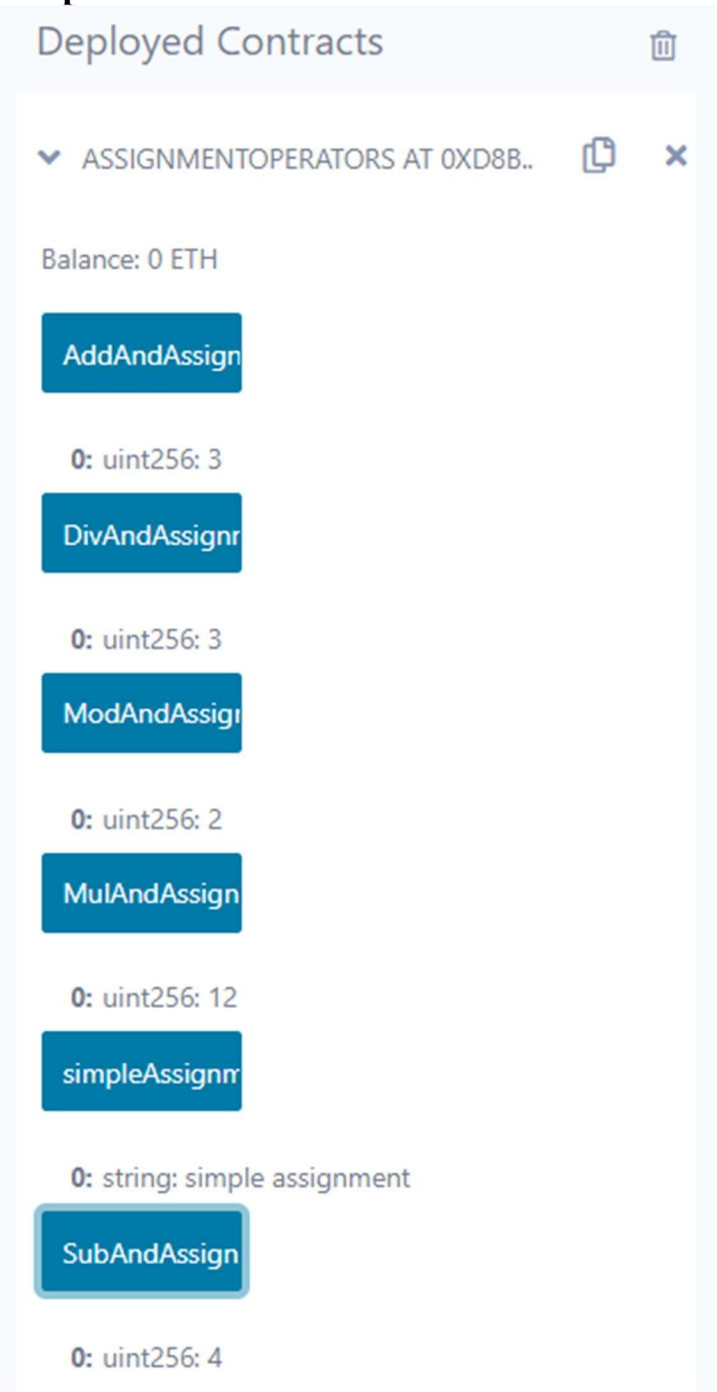
4c) Aim: Solidity program to demonstrate Assignment operators.

Code:

4_c_AssignmentOperator.sol

```
// SPDX-License-Identifier: GPL-3.0

pragma solidity >=0.8.2 <0.9.0;

contract AssignmentOperators {

    //  =
    function simpleAssignment() public pure returns (string memory) {    infinite gas
        string memory result = "simple assignment";
        return result;
    }

    // +=
    function AddAndAssignment() public pure returns (uint) {    infinite gas
        uint result = 1;
        result +=2;
        return result;
    }

    // -=
    function SubAndAssignment() public pure returns (uint) {    infinite gas
        uint result = 6;
        result -=2;
        return result;
    }

    // *=
    function MulAndAssignment() public pure returns (uint) {    infinite gas
        uint result = 6;
        result *=2;
        return result;
    }

    // /=
    function DivAndAssignment() public pure returns (uint) {    infinite gas
        uint result = 6;
        result /=2;
        return result;
    }

    // %=
    function ModAndAssignment() public pure returns (uint) {    infinite gas
        uint result = 6;
        result %=4;
        return result;
    }
}
```

Output:


Deployed Contracts
ASSIGNMENTOPERATORS AT 0XD8B..
Balance: 0 ETH
AddAndAssign
0: uint256: 3
DivAndAssignr
0: uint256: 3
ModAndAssigı
0: uint256: 2
MulAndAssign
0: uint256: 12
simpleAssignm
0: string: simple assignment
SubAndAssign
0: uint256: 4

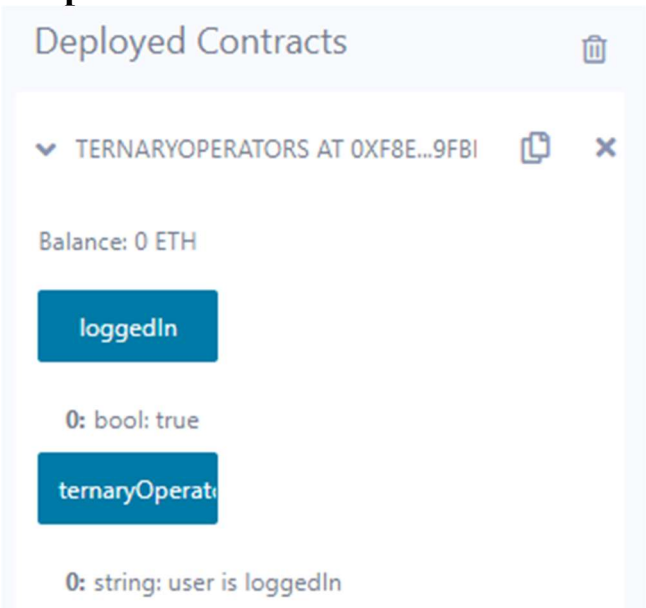
4d) Aim: Solidity program to demonstrate Ternary operators.

Code:

4_c_AssignmentOperator.sol | 4_d_ternary.sol

```
// SPDX-License-Identifier: GPL-3.0

pragma solidity >=0.8.2 <0.9.0;

contract TernaryOperators {

    bool public loggedIn = true;

    function ternaryOperator() public view returns (string memory) {    infinite gas
        string memory result = (loggedIn) ? "user is loggedIn" : "user is not loggedIn";
        return result;
    }
}
```

Output:



4e) Aim: Solidity program to demonstrate Bitwise operators.

Code:

4_e_bitwise.sol

```
// SPDX-License-Identifier: GPL-3.0

pragma solidity >=0.8.2 <0.9.0;

contract TernaryOperators {

    int public a = 10;
    int public b = 12;

    // & (Bitwise AND)
    int public OfAND = a & b;

    // | (BitWise OR)
    int public OfOR = a | b;

    // ^ (Bitwise XOR)
    int public OfXOR = a ^ b;

    // ~ (Bitwise Not)
    int public OfNOT = (~a);

    // << (Left Shift)
    int public OfLEFTSHIFT = a << 2;

    // >> (Right Shift)
    int public OfRIGHTSHIFT = a >> 2;

}
```

Output:

Deployed Contracts

TERNARYOPERATORS AT 0XD2A...FD005 (M

Balance: 0 ETH

a

0: int256: 10

b

0: int256: 12

OfAND

0: int256: 8

OfLEFTSHIFT

0: int256: 40

OfNOT

0: int256: -11

OfOR

0: int256: 14

OfRIGHTSHIFT

0: int256: 2

OfXOR

0: int256: 6

Practical No 5

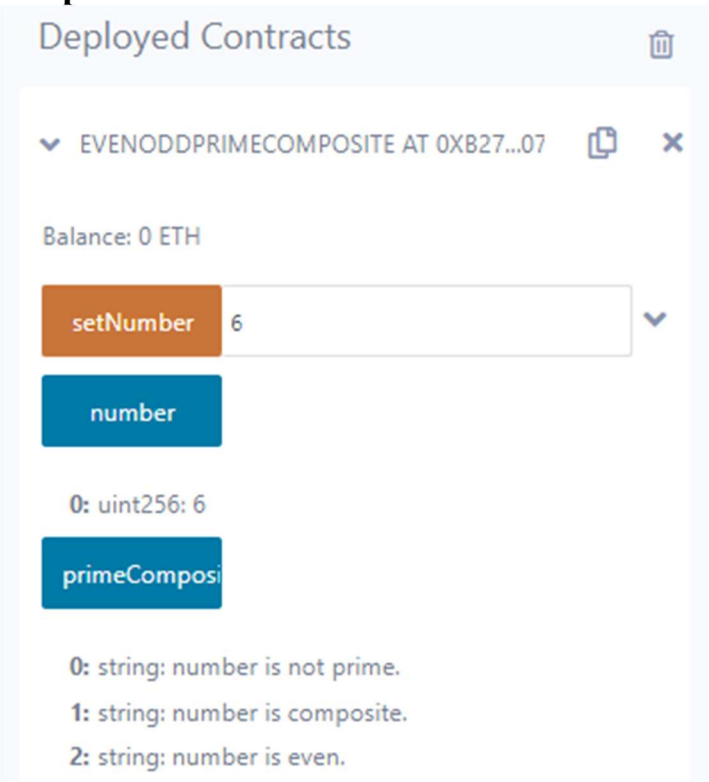
Aim: Write a solidity program to find if the number is odd or even, prime or composite.

Code:

4_e_bitwise.sol | 5_evenOddPrimeComposite.sol

```
// SPDX-License-Identifier: GPL-3.0

pragma solidity >=0.8.2 <0.9.0;

contract evenOddPrimeComposite {

    uint public number;

    function setNumber(uint _number) public {    22498 gas
        number = _number;
    }

    function primeCompositeEvenOddChecker () public view returns(string memory, string memory, string memory) {

        string memory resultOfPrime;
        string memory resultOfComposite;
        string memory resultOfEvenOdd;

        if(number <=1) {
            resultOfPrime = "not a prime number.";
            resultOfComposite = "not a composite number.";
        }
        for(uint i =2;i<number/2;i++) {
            if(number % i == 0) {
                resultOfPrime = "number is not prime.";
                resultOfComposite = "number is composite.";
            }
        }

        if(number %2 == 0) {
            resultOfEvenOdd = "number is even.";
        } else {
            resultOfEvenOdd = "number is odd.";
        }
        return (resultOfPrime, resultOfComposite, resultOfEvenOdd);
    }
}

```

Output:


Deployed Contracts
EVENODDPRIMECOMPOSITE AT 0XB27...07
Balance: 0 ETH
setNumber
6
number
0: uint256: 6
primeComposi
0: string: number is not prime.
1: string: number is composite.
2: string: number is even.

Practical No 6

Aim: Write a solidity program to find modulus of addition of two numbers with DD and modulus of multiplication of two numbers with MM where the two numbers are achieved by performing AND operation of DD and MM of your date of birth and OR operation of YY and YY from year of birth.

Code:

6_birthdate.sol

```
pragma solidity >=0.8.2 <0.9.0;

contract Birthdate {
    uint DD = 21;
    uint MM = 4;
    uint YY = 19;
    uint YYM = 99;

    uint public number1 = DD & MM;
    uint public number2 = YY | YYM;

    uint public modOfADDOfTwoNumbers = addmod(number1, number2, DD);
    uint public modOfMULOfTwoNumbers = addmod(number1, number2, MM);
}
```

Output:

Deployed Contracts

BIRTHDATE AT 0X9D8...A5692 (MEMORY)

Balance: 0 ETH

modOfADDOf

0: uint256: 14

modOfMULOf

0: uint256: 0

number1

0: uint256: 4

number2

0: uint256: 115

Practical No 7

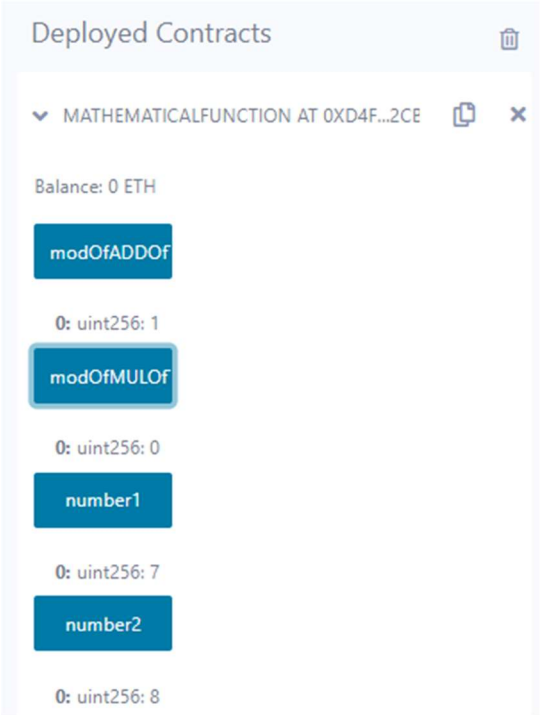
7a) Aim: Create a smart contract to demonstrate Mathematical function.

Code:

7_a_mathematicalFunctions.sol

```
// SPDX-License-Identifier: GPL-3.0

pragma solidity >=0.8.2 <0.9.0;

contract MathematicalFunction {

    uint public number1 = 7;
    uint public number2 = 8;

    uint public modOfADDOfTwoNumbers = addmod(number1, number2, 2);
    uint public modOfMULOfTwoNumbers = mulmod(number1, number2, 4);
}

```

Output:



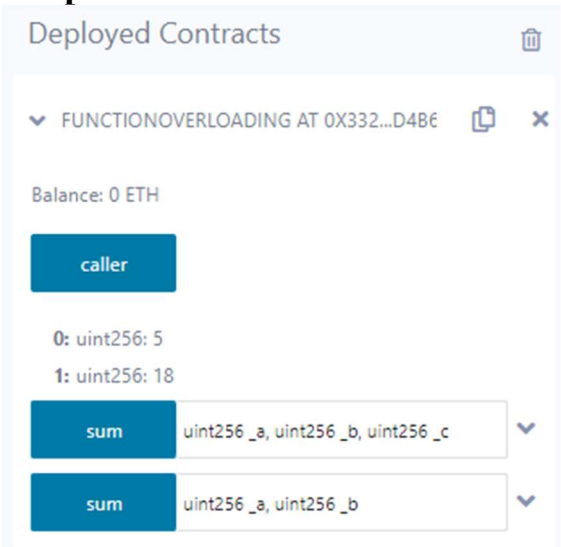
7b) Aim: Create a smart contract to demonstrate Function overloading.

Code:

7_b_functionOverloading.sol

```
// SPDX-License-Identifier: GPL-3.0

pragma solidity >=0.8.2 <0.9.0;

contract functionOverloading {

    function sum(uint _a, uint _b) public pure returns(uint) {    infinite gas
        uint result = _a + _b;
        return result;
    }

    function sum(uint _a, uint _b, uint _c) public pure returns(uint) {    in
        uint result = _a + _b + _c;
        return result;
    }

    function caller() public pure returns(uint, uint) {    infinite gas
        return (sum(2,3), sum(5,6,7));
    }
}
```

Output:



Practical 8

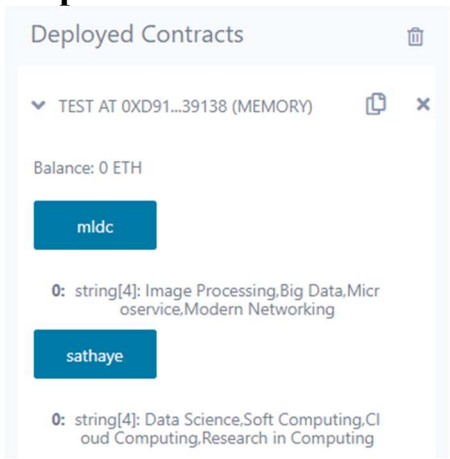
8a) Aim: Create a smart contract to show the implementation of Interface.

Code:

8_a_interfaceImplentation.sol

```
// SPDX-License-Identifier: GPL-3.0

pragma solidity >=0.8.2 <0.9.0;

/* Consider university of mumbai as interface where as to provide subject electives to two colleges  sathaye
   and mldc(methods inside interface),
   create a smart contract to provide the list of sem-1 subjects in the form of array to sathaye and sem-2 subjects to mldc.
*/


interface MumbaiUniversity {
   function sathaye() external pure returns (string[4] memory);   - gas
   function mldc() external pure returns (string[4] memory);   - gas
}

contract Test is MumbaiUniversity {

   function sathaye() public pure override returns (string[4] memory) {   infinite gas
      string[4] memory sem_1 = ["Data Science", "Soft Computing","Cloud Computing","Research in Computing"];
      return sem_1;
   }

   function mldc() public pure override returns (string[4] memory) {   infinite gas
      string[4] memory sem_2 = ["Image Processing", "Big Data", "Microservice", "Modern Networking"];
      return sem_2;
   }
}

```

Output:



8b) Aim: Create a smart contract to show the implementation of Inheritance.

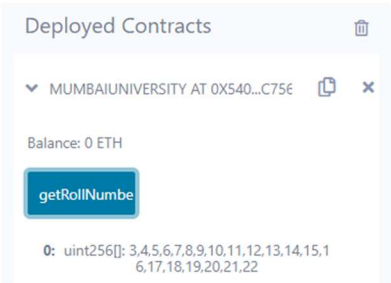
Code:

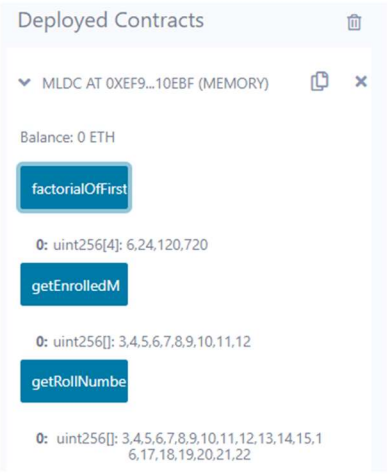
8_b_inheritanceImplementation.sol

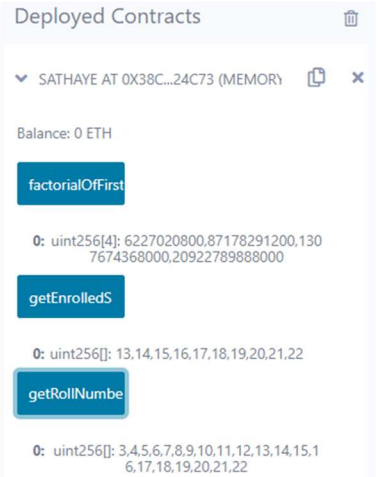
```
// SPDX-License-Identifier: GPL-3.0

pragma solidity >=0.8.2 <0.9.0;

/*
Consider university of mumbai as the base class allocating array of roll numbers to two colleges
mldc and sathaye(methods in class).
Where enrollment will happen and hence create a smart contract to place the first ten roll numbers
in mldc contract and last ten roll numbers in sathaye contract, also find the factorial of first four numbers in
each contract.
*/

contract MumbaiUniversity {
    uint256[] internal rollNumbers;

    constructor() {     infinite gas 12600 gas
        for (uint256 i = 0; i < 20; i++) {
            rollNumbers.push(i + 3);
        }
    }
}

contract MLDC is MumbaiUniversity {
    uint256[] private enrolledToMLDC;

    constructor() {     infinite gas 219000 gas
        for (uint256 i = 0; i < 10; i++) {
            enrolledToMLDC.push(rollNumbers[i]);
        }
    }

    function getEnrolledM() public view returns (uint256[] memory) {     infinite gas
        return enrolledToMLDC;
    }

    function factorialOfFirstFour() public view returns (uint256[4] memory) {     infinite gas

        uint256[4] memory factorials;
        for (uint256 i = 0; i < 4; i++) {
            uint256 num = enrolledToMLDC[i];
            uint256 factorial = 1;
            for (uint256 j = 2; j <= num; j++) {
                factorial *= j;
            }
            factorials[i] = factorial;
        }
        return factorials;
    }
}
```

```
contract Sathaye is MumbaiUniversity {
    uint256[] private enrolledToSATHAYE;

    constructor() {      infinite gas 219000 gas
        for (uint256 i = 10; i < rollNumbers.length; i++) {
            enrolledToSATHAYE.push(rollNumbers[i]);
        }
    }

    function getEnrolledS() public view returns (uint256[] memory) {      infinite gas
        return enrolledToSATHAYE;
    }

    function factorialOfFirstFour() public view returns (uint256[4] memory) {      infinite gas

        uint256[4] memory factorials;
        for (uint256 i = 0; i < 4; i++) {
            uint256 num = enrolledToSATHAYE[i];
            uint256 factorial = 1;
            for (uint256 j = 2; j <= num; j++) {
                factorial *= j;
            }
            factorials[i] = factorial;
        }
        return factorials;
    }
}
```

Output:







Practical No 9

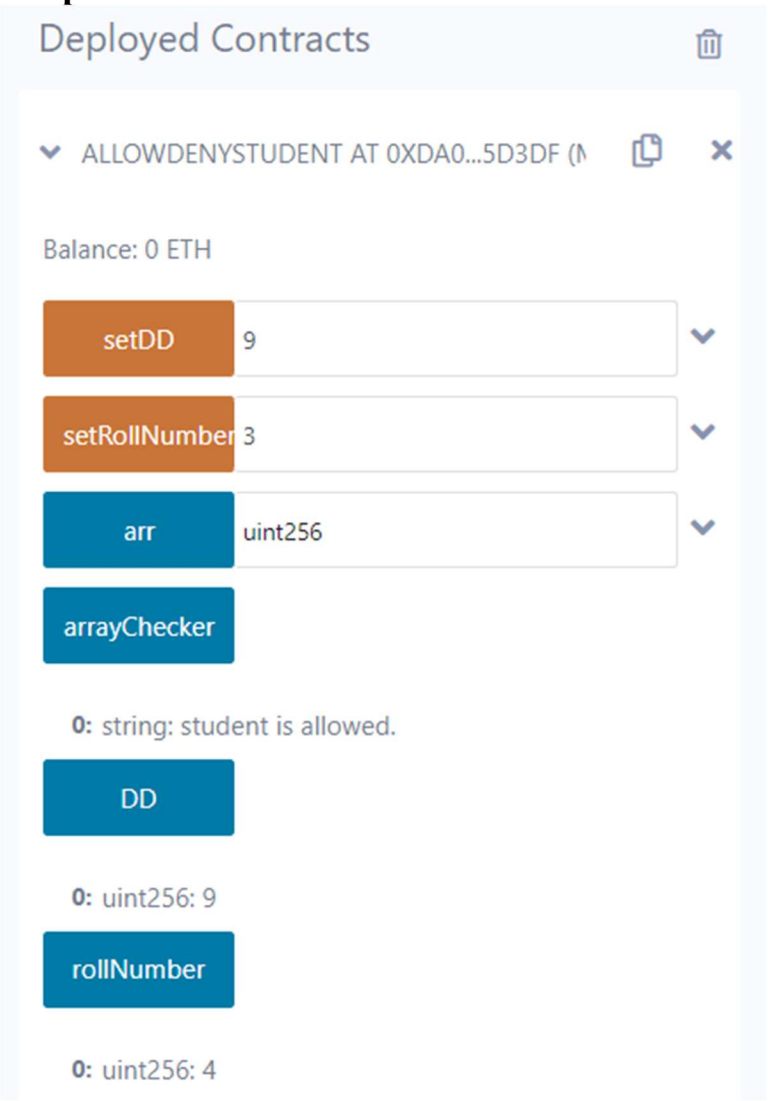
Aim: Write a solidity program to create an array of roll numbers and then create a smart contract where it checks the value of the roll number and perform AND operation with today's date and if the result is even then allow the student else deny (DD part only).

Code:

9_allowDenyStudent.sol

```
// SPDX-License-Identifier: GPL-3.0

pragma solidity >=0.8.2 <0.9.0;

contract allowDenyStudent {
    uint256[] public arr = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10];
    uint256 public rollNumber;
    uint256 public DD;

    function setRollNumber(uint256 _rollNumber) public {    26800 gas
        rollNumber = arr[_rollNumber];
    }

    function setDD(uint256 _DD) public {    22498 gas
        DD = _DD;
    }

    function arrayChecker() public view returns (string memory) {
        string memory finalOutput;

        uint256 operation = rollNumber & DD;

        if (operation % 2 == 0) {
            finalOutput = "student is allowed.";
        } else {
            finalOutput = "student is denied.";
        }

        return finalOutput;
    }
}
```

Output:


Deployed Contracts
ALLOWDENYSTUDENT AT 0XDA0...5D3DF (M
Balance: 0 ETH
setDD
9
setRollNumber
3
arr
uint256
arrayChecker
0: string: student is allowed.
DD
0: uint256: 9
rollNumber
0: uint256: 4

Practical No 10

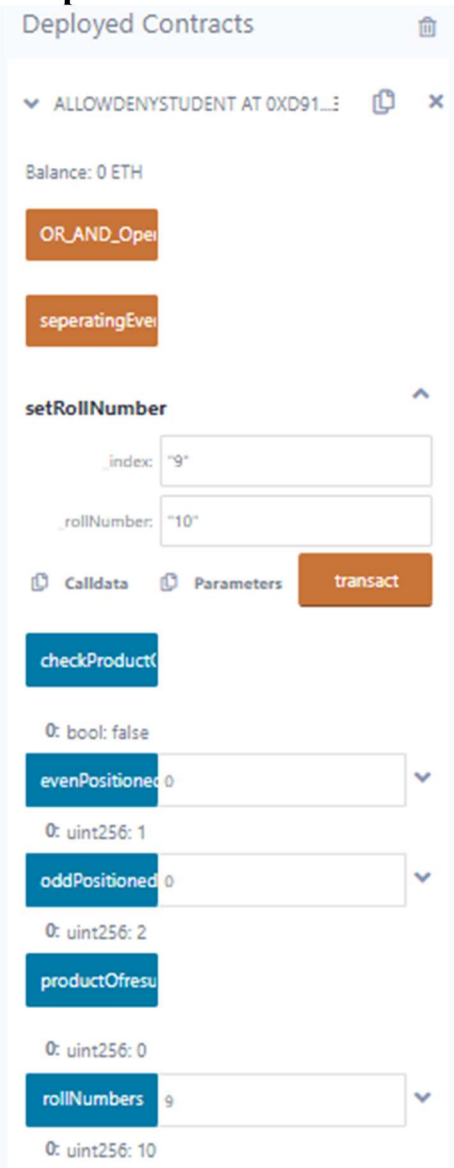
Aim: Write a solidity program to find the sum of an array of ten numbers using loop the numbers are expected to be taken from the user, create a smart contract to find the AND operation of odd positioned numbers and OR operation of even positioned numbers including 0th index. Hence find the product of the results and also identify whether the result is the part of array or not.

Code:

Home | 10_pract.sol

```
// SPDX-License-Identifier: GPL-3.0

pragma solidity >=0.8.2 <0.9.0;

contract allowDenyStudent {

    uint256[10] public rollNumbers;

    function setRollNumber(uint _index ,uint _rollNumber) public {    infinite gas
        // push() method is not available on fixed size array.
        rollNumbers[_index] = (_rollNumber);
    }

    function seperatingEvenOdd() public {    infinite gas

    for(uint i=0;i<rollNumbers.length;i++) {

        if(i %2 == 0) {
            evenPositioned.push(rollNumbers[i]);
        } else {
            oddPositioned.push(rollNumbers[i]);
        }
    }
    }

    uint[] public evenPositioned;
    uint[] public oddPositioned;

    function OR_AND_Operation() public {    infinite gas

        uint resultOfOR = evenPositioned[0];
        for(uint i=1;i<evenPositioned.length;i++) {
            resultOfOR = resultOfOR | evenPositioned[i];
        }

        uint resultOfAND = oddPositioned[0];
        for(uint i=1;i<oddPositioned.length;i++) {
            resultOfAND = resultOfAND & oddPositioned[i];
        }
        productOfresults = resultOfOR * resultOfAND;
    }

    uint256 public productOfresults;

    function checkProductOfResults() public view returns(bool) {    infinite gas
        bool result;
        for(uint i=0; i<rollNumbers.length; i++) {

            result = (productOfresults == rollNumbers[i]) ? true : false;
        }
        return result;
    }
}
```

Output:


Deployed Contracts
ALLOWDENYSTUDENT AT 0XD91...
Balance: 0 ETH
OR_AND_Oper
seperatingEve
setRollNumber
_index: "9"
_rollNumber: "10"
Calldata
Parameters
transact
checkProduct(
0: bool: false
evenPositioned 0
0: uint256: 1
oddPositioned 0
0: uint256: 2
productOfresu
0: uint256: 0
rollNumbers 9
0: uint256: 10